Advanced OO Design Principles

Unit 2 – Composition vs. Inheritance

Pratian Technologies (India) Pvt. Ltd.
www.pratian.com

PRATIAN
TECHNOLOGIES

PRATIAN TECHNOLOGIES

Topics

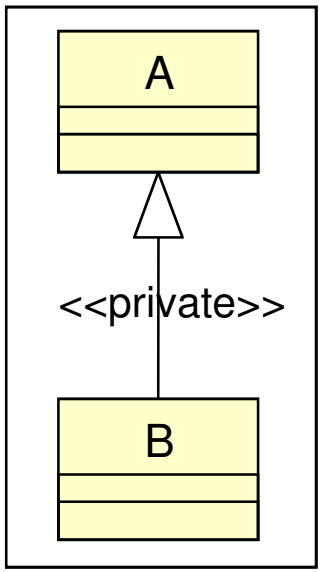
- Inheritance and maintainability
  - Private Inheritance vs. Composition
  - Multiple Inheritance vs. Composition
- Understanding the ‘Has-a’ relationship
  - Exercises
  - Case Studies
- Types of Collaboration
  - Auto Collaboration
  - Base-Derived Collaboration
  - Sibling Collaboration
  - Peer to peer Collaboration
- Prefer ‘Composition over Inheritance

Inheritance & Maintainability

PRATIAN TECHNOLOGIES

Remodeling Private Inheritance

- **Consider the following case**



- B needs to provide some implementations for its methods exposed to the clients. The developer of class B discovers that some implementation is readily available in class A and hence can be reused. The problem however is that the reuse of implementation should be transparent to the clients. The implementations used by B from A cannot be exposed on the class B. It needs to keep the implementations localized to the class B only.
- Private Inheritance is a good candidate to solve this problem in C++

- **How would you re-model this?**

PRATIAN TECHNOLOGIES

Remodeling Private Inheritance

- Design the solution to the problem
- Understand the reasons why C# or Java does not allow private inheritance
- How would you suggest that this solution is more maintainable than private inheritance
- Understand that reuse need not be achieved only through Inheritance

PRATIAN TECHNOLOGIES

Problems with Multiple Inheritance (MI)

- The two types of Inheritance in a OO language are
  - Implementation Inheritance (Generalization)
  - Interface Inheritance (Realization)
- Eliminate Multiple Inheritance (MI) between classes
- Possible to have multiple inheritance between interfaces
- Possible for a class to realize multiple interfaces

- Multiple Implementation Inheritance is a problem
- Multiple Interface Inheritance is possible

PRATIAN TECHNOLOGIES

Why is MI bad?

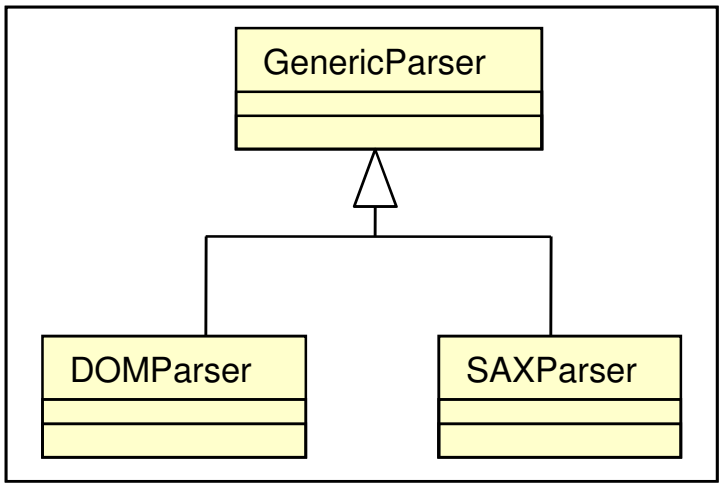
- Redundancy in implementation reuse
- Replication of state
- Ambiguity in invoking methods of the Base classes
- Virtual Base classes have overheads of '**Virtual ness**'
- Simply put, **Not maintainable**

- Hence a good design,
  - allows multiple
    - Interface Inheritance
  - disallows multiple
    - Implementation Inheritance

PRATIAN TECHNOLOGIES

Multiple Inheritance (MI)

- Consider the following scenario



How would you create a HybridParser, which needs to reuse the implementations of both DOMParser and SAXParser?

Understanding ‘Has-a’ relationship

PRATIAN TECHNOLOGIES

Understanding the 'Has-a' relationship

- Ideally 'has-a' can be represented as
  - Association
  - Aggregation
  - Composition
- For the purpose of simplicity, we will represent 'has-a' as aggregation
- Inheritance Vs Aggregation
  - Should be interpreted as 'is-a' Vs 'has-a' in our subsequent discussions to understand the 'has-a' relationship

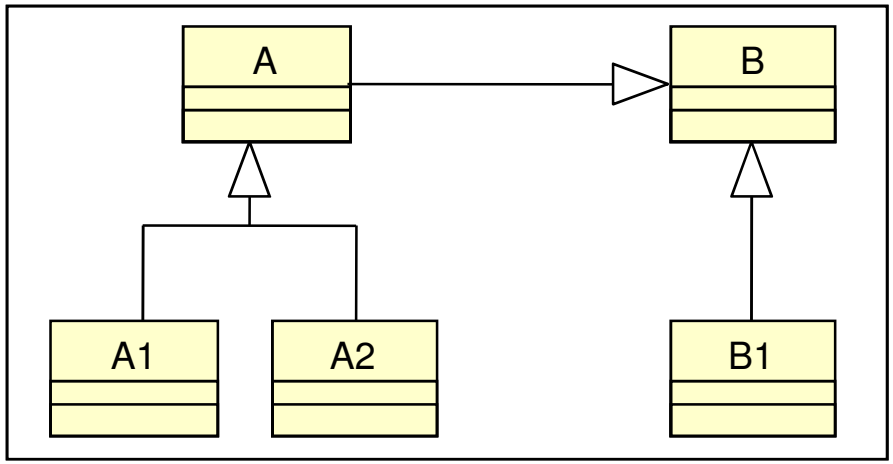
PRATIAN TECHNOLOGIES

Problem 1

- Consider the above diagram
  - Name at least three different ways of storing an instance of B in an instance of A
- The mechanism of storing the instance (the implementation code) is however not significant by itself during design. Interpreting the impact of this design and creating abstractions are more important

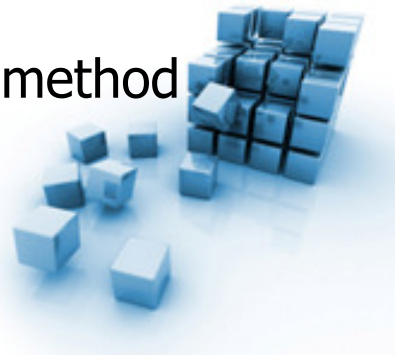
PRATIAN TECHNOLOGIES

Problem 2



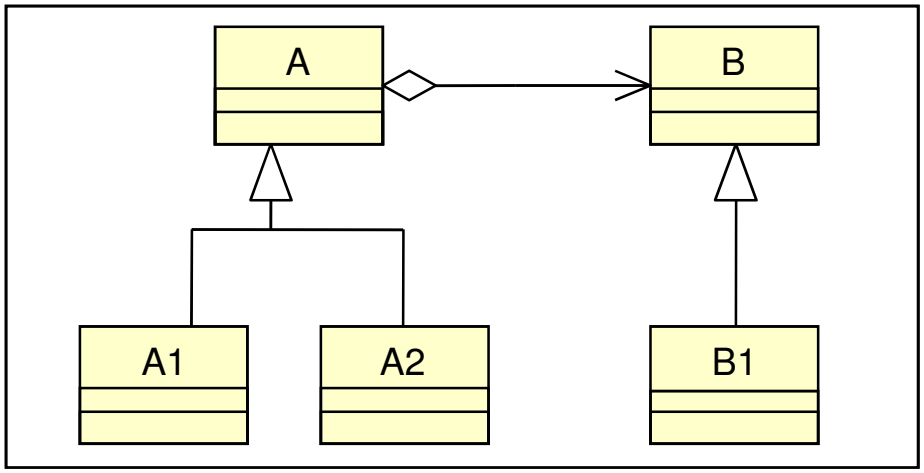
Consider the above diagram. Which among the following statements are true?

1. 'A' can use the implementation of B
2. 'A' can use the implementation of B1
3. 'A1' can use the implementation of B
4. 'A' can behave like B
5. 'A2' cannot behave like B
6. Objects of 'A', 'A1', 'A2', 'B1' can be passed to a method which receives a reference to B



PRATIAN TECHNOLOGIES

Problem 3

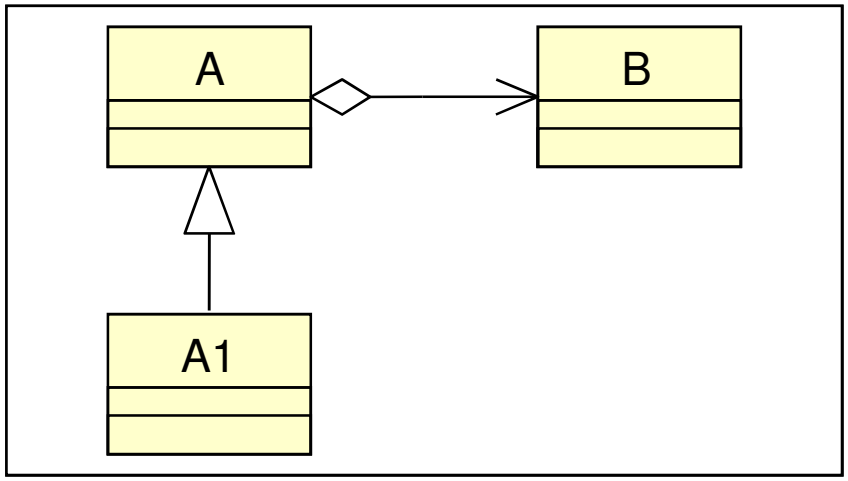


Consider the above diagram. Which among the following statements are true?

1. 'A' can use the implementation of B
2. 'A' can use the implementation of B1
3. 'A1' can use the implementation of B
4. 'A2' can use the implementation of B1
5. 'A' can behave like B
6. Objects of 'A', 'A1', 'A2' can be passed to a method which receives a reference to B

PRATIAN TECHNOLOGIES

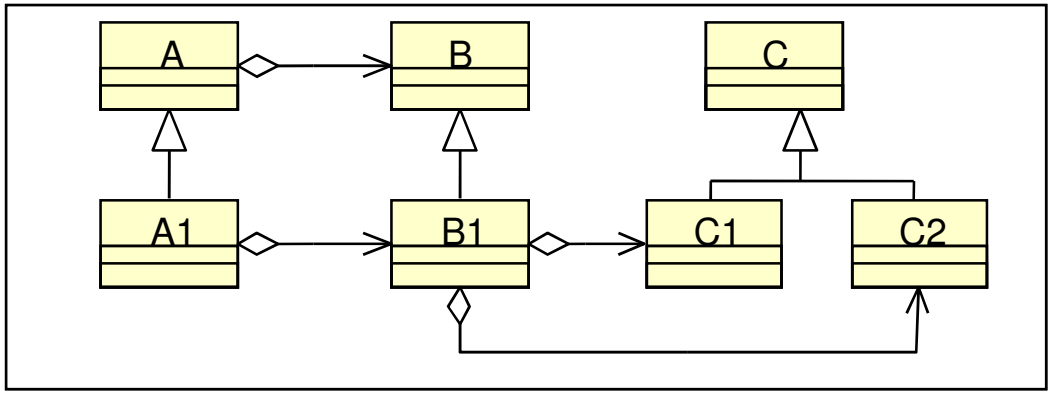
Problem 4



- Consider the above diagram
  - Name at least two different ways of storing an instance of **B** in an instance of **A1**

PRATIAN TECHNOLOGIES

Problem 5



- Consider the above diagram. Given that
  - A has B
  - A1 has B
  - A1 has B1
  - B1 has C1
  - B1 has C2
  - B cannot have C
  - B1 cannot have C
- Refine the above diagram to achieve the same. Keep in mind maintainability when C3, C4, etc. may need to be accommodated in B1.

Types of Collaboration

PRATIAN TECHNOLOGIES

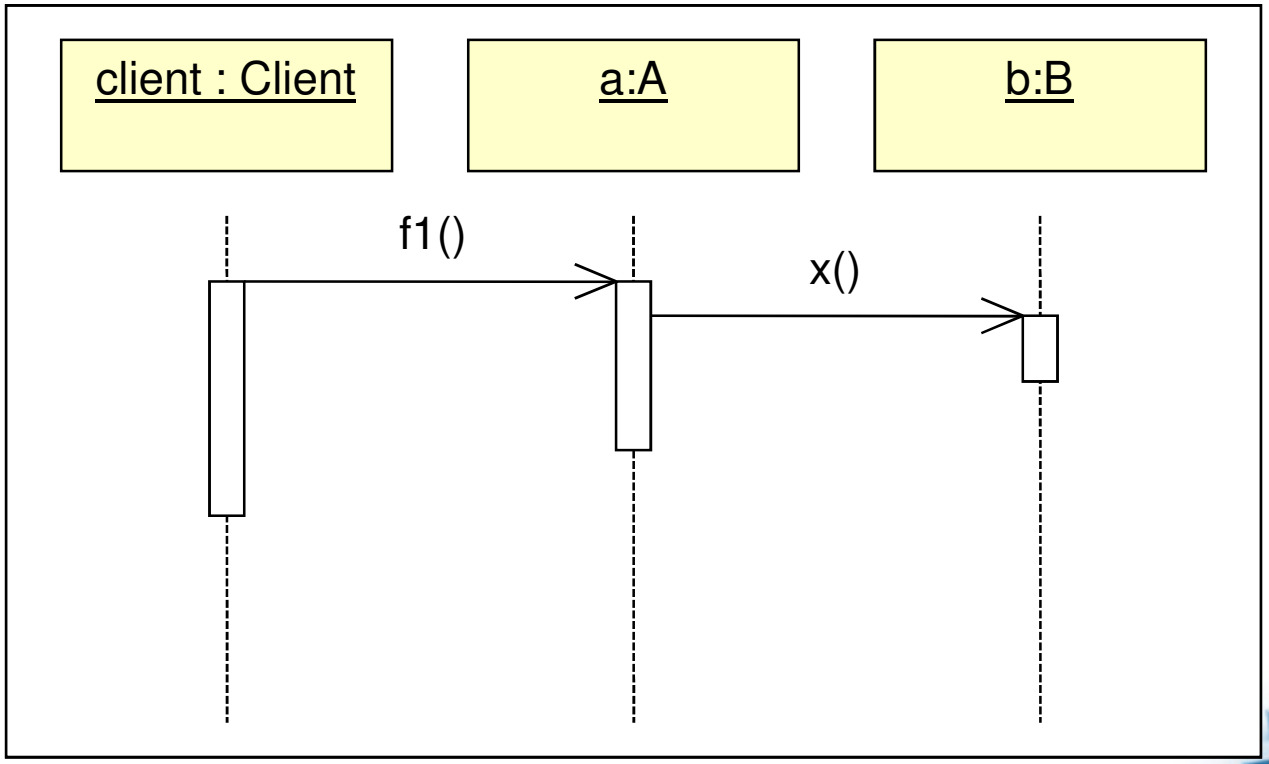
Object Collaborations

- 'The objects within a program must collaborate; otherwise, the program would consist of only one big object that does everything.'

PRATIAN TECHNOLOGIES

Collaboration

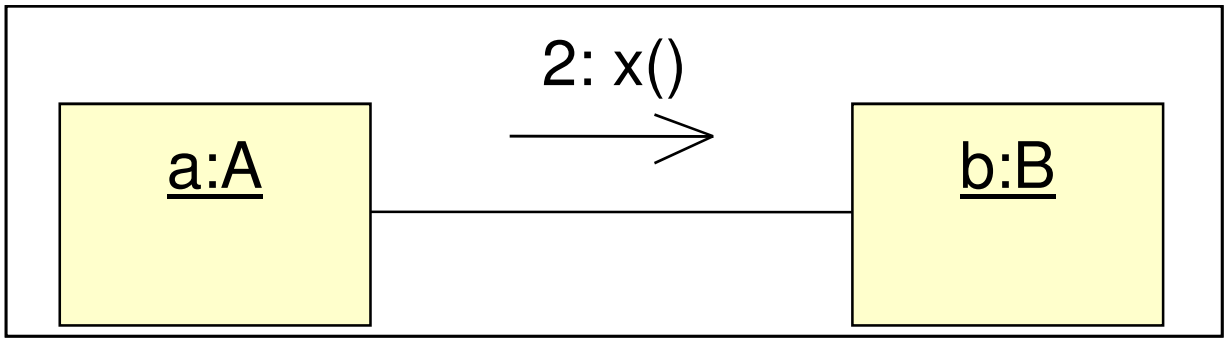
- A collaboration occurs every time two or more objects interact
- A relationship between the two objects is hence established
- High Cohesion and low coupling desired



PRATIAN TECHNOLOGIES

Object Collaborations

- What are the different ways through which the method ‘x’ can be invoked from within the class ‘A’ ?



- Is it possible to model a collaboration through any relationship – is-a, has-a or uses?

PRATIAN TECHNOLOGIES

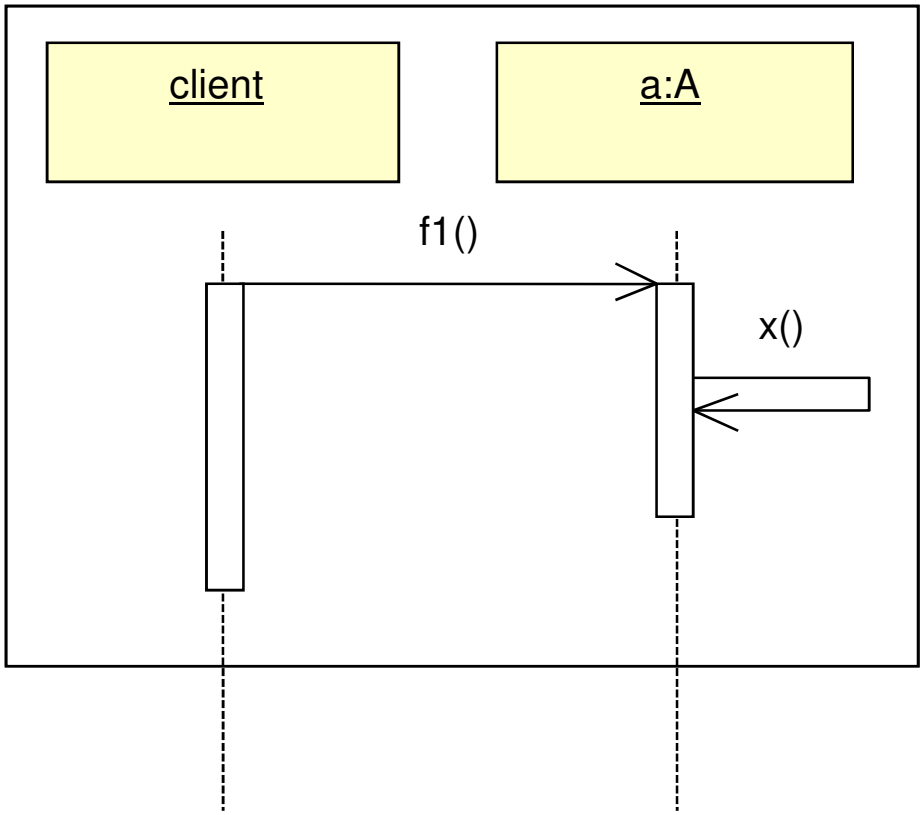
Collaboration types

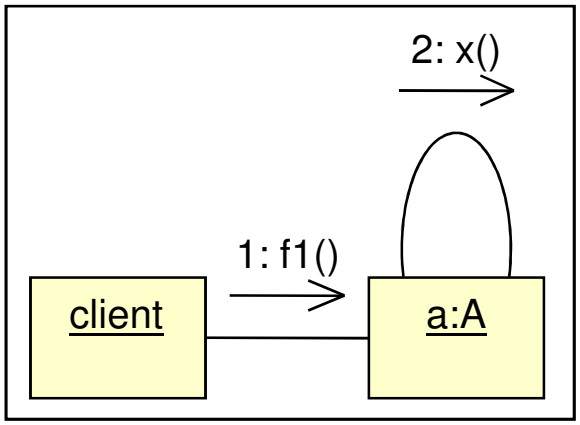
- Auto collaboration
  - A message to self
- Base-derived collaboration
- Sibling collaboration
  - Derived classes of the same class hierarchy collaborate
- Peer to peer collaboration
  - Two unrelated objects collaborate

PRATIAN TECHNOLOGIES

Auto Collaboration

- A message to self





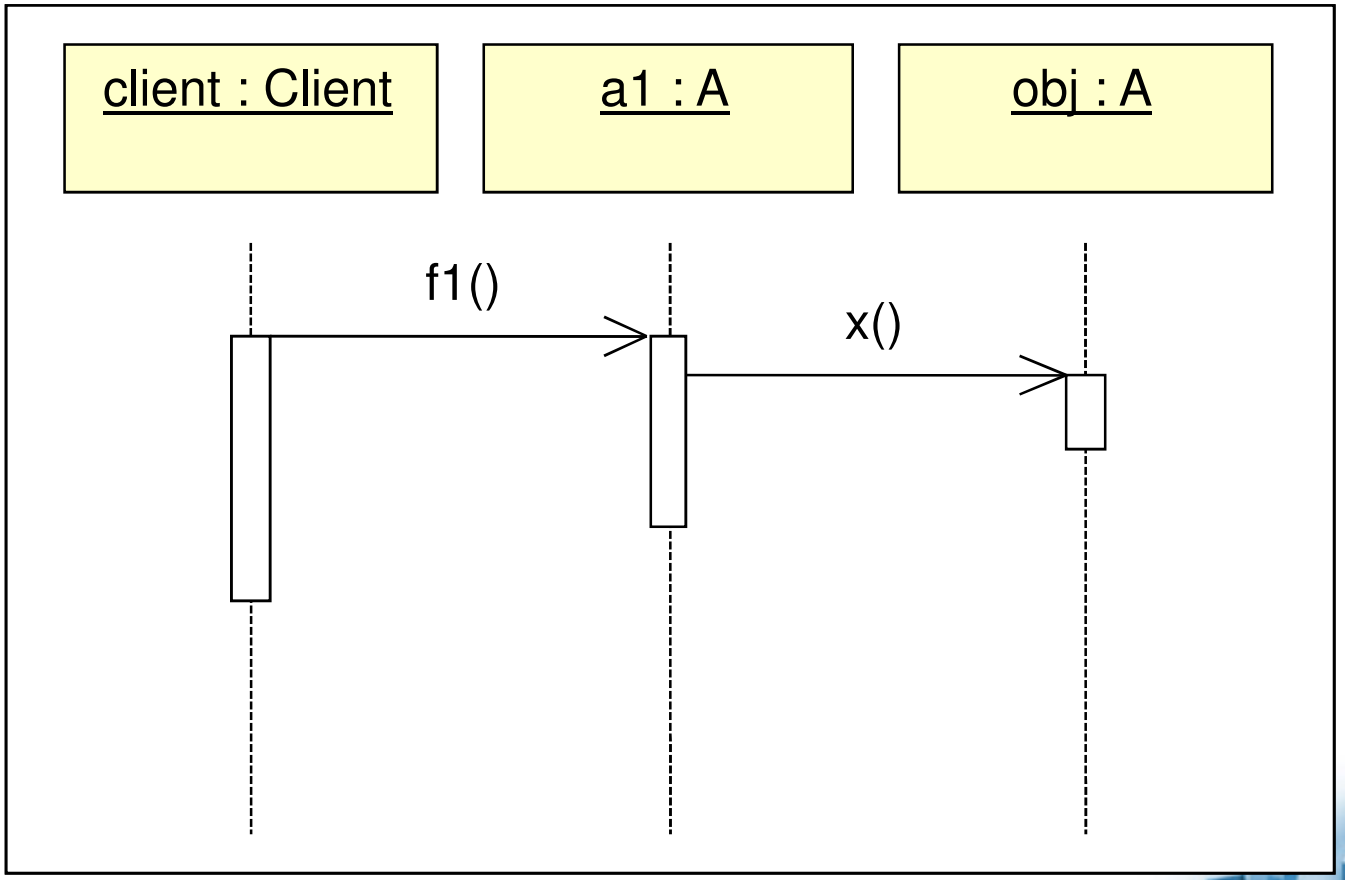

```
class A {
        public void f1() {
                this.x();
        }
        private void x() { }
}
```

- Auto Collaboration is a good pointer to private methods

PRATIAN TECHNOLOGIES

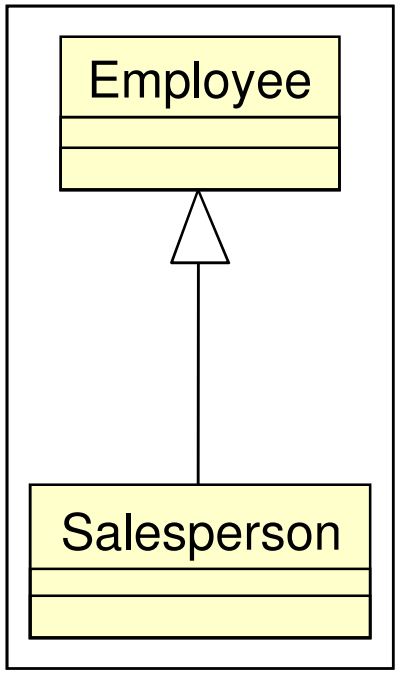
Auto Collaboration

- Where will you capture the message passed from object a1 to 'obj'?



PRATIAN TECHNOLOGIES

Base-Derived Collaboration



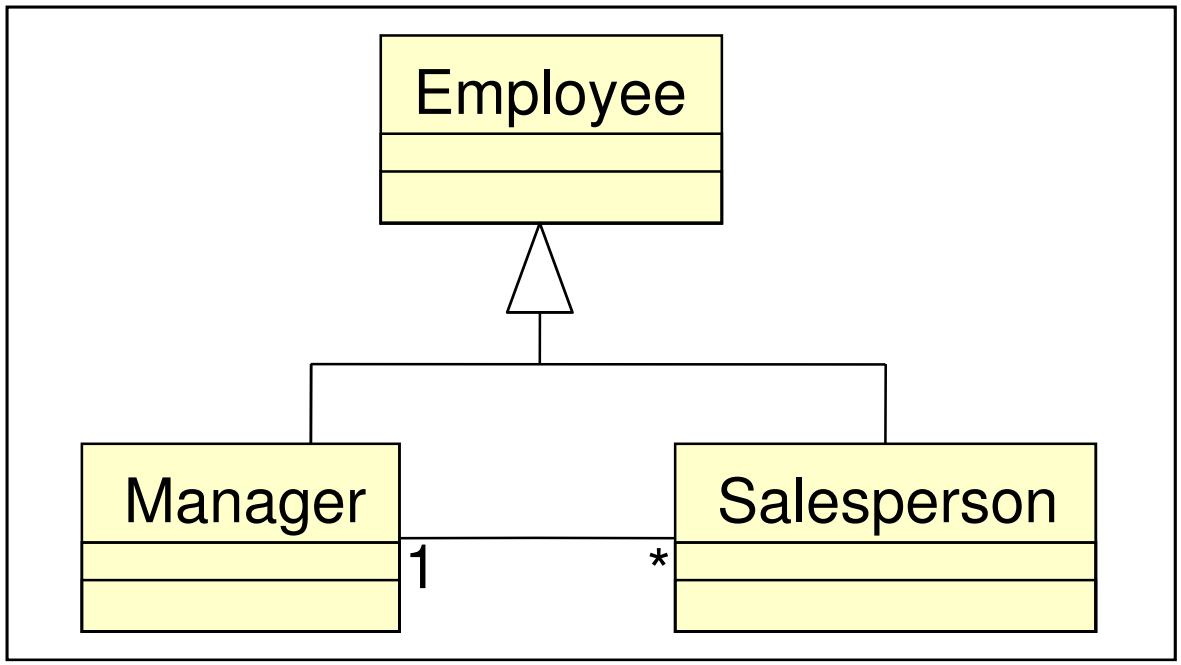

```
class Employee {
    public double getSalary() {
            double sal =
                SalCalcFactory.getInstance(dtEmp).findSalary(salary);
            return sal;
    }
}
class Salesperson extends Employee {
    public double getSalary() {
            double sal = super.getSalary() +
                SalCalcFactory.getInstance(dtEmp).findIncentives(sales);
            return sal;
            }
}
```

An interaction between super class and sub class

PRATIAN TECHNOLOGIES

Sibling Collaboration

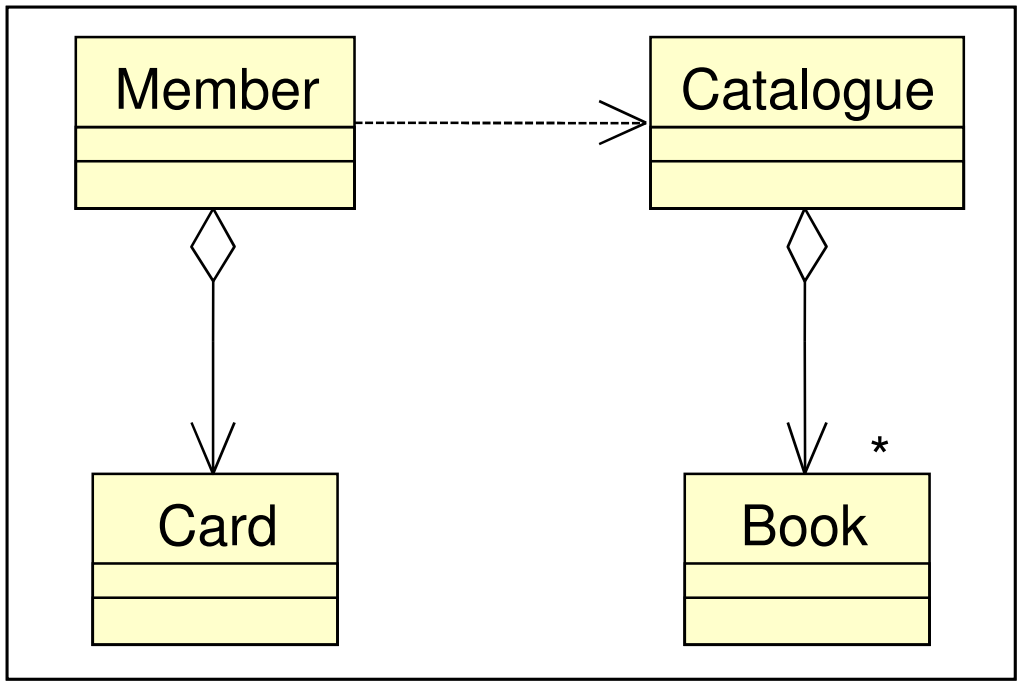
- Derived classes of the same class hierarchy collaborate



PRATIAN TECHNOLOGIES

Peer To Peer Collaboration

- An interaction between unrelated classes



PRATIAN TECHNOLOGIES

Peer To Peer Collaboration

```
class Member
{
    Card card;
    void requestBook(Catalogue catalogue)
    {
      catalogue.viewBookByTitle("Java Design
                                 Patterns");
        //
    }
}
```

<< Collaboration >>

```
class Catalogue
{
    private Book[] books;
    private getBookByTitle(String title)
    {
            Book book;
            for(int i=0;i<numOBooks;i++)
                    // find the book
            return book;
    }
    void viewBookByTitle(String title)
    {
            Book book =
getBookByTitle(title);
            book.showBookDetails();
    }
}
```

PRATIAN TECHNOLOGIES

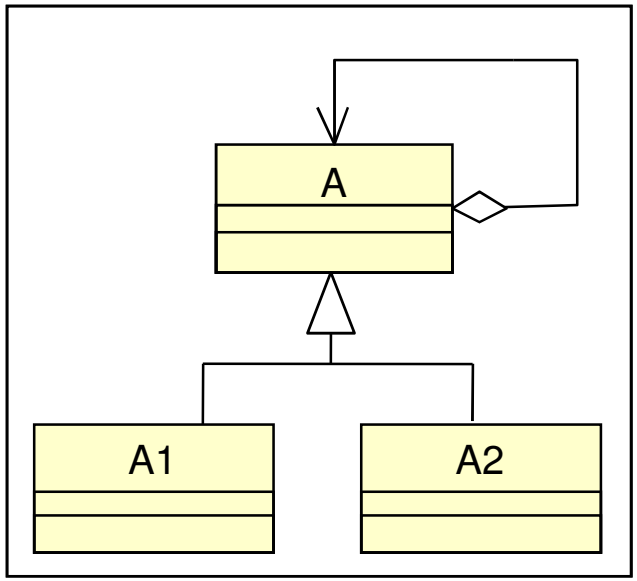
Collaboration & Relationships

- Let us now identify the most appropriate relationship for a type of collaboration

Collaboration	is-a	has-a
Auto Base Derived Sibling Peer to peer		

PRATIAN TECHNOLOGIES

Auto Collaboration



- Which of the following statements are true?

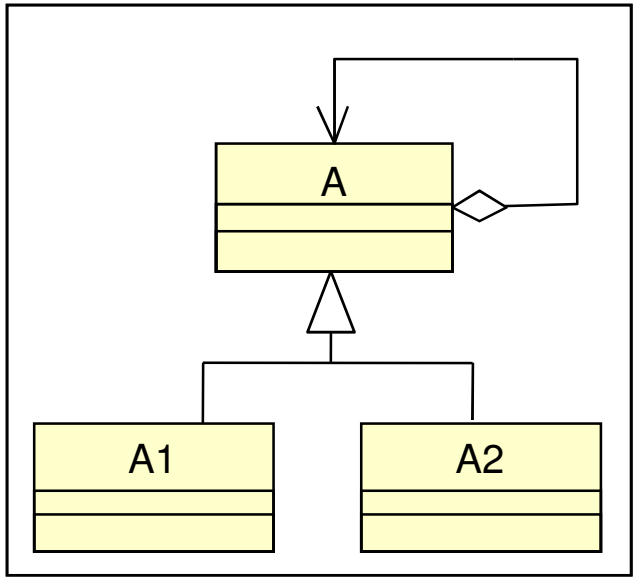
A can have A
A can have A1
A can have A2

A1 can have A
A1 can have A1
A1 can have A2

A2 can have A
A2 can have A1
A2 can have A2

PRATIAN TECHNOLOGIES

Auto Collaboration



- Based on your findings, state which of the following statements are possible:
- Structural Auto Collaboration
  - Allows bi-directional Base-Derived collaboration
  - Allows Sibling collaboration
  - Allows Auto collaboration for the sub classes in the hierarchy

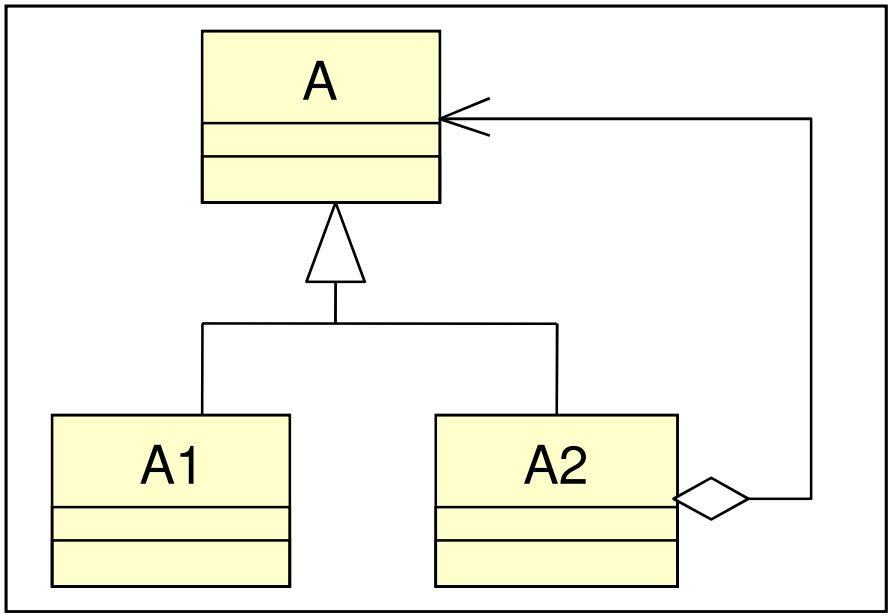
PRATIAN TECHNOLOGIES

Auto Collaboration

Collaboration	is-a	has-a
Auto Base Derived Sibling Peer to peer	N	Y

PRATIAN TECHNOLOGIES

Base Derived Collaboration



- Which of the following statements are true?

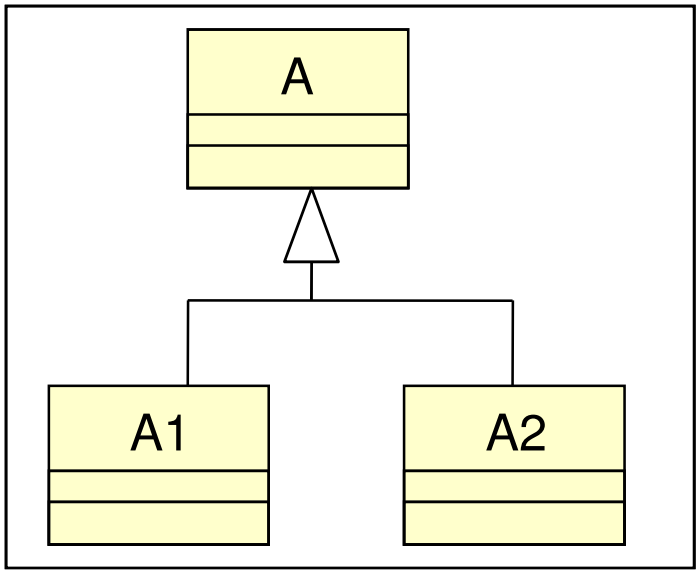
A can have A
A can have A1
A can have A2

A1 can have A
A1 can have A1
A1 can have A2

A2 can have A
A2 can have A1
A2 can have A2

PRATIAN TECHNOLOGIES

Base Derived Collaboration



- Given that the following relationships need to be achieved, how would you model the above classes

A1 can have A
A1 can have A1
A1 can have A2

A2 can have A
A2 can have A1
A2 can have A2

A cannot have A
A cannot have A1
A cannot have A2

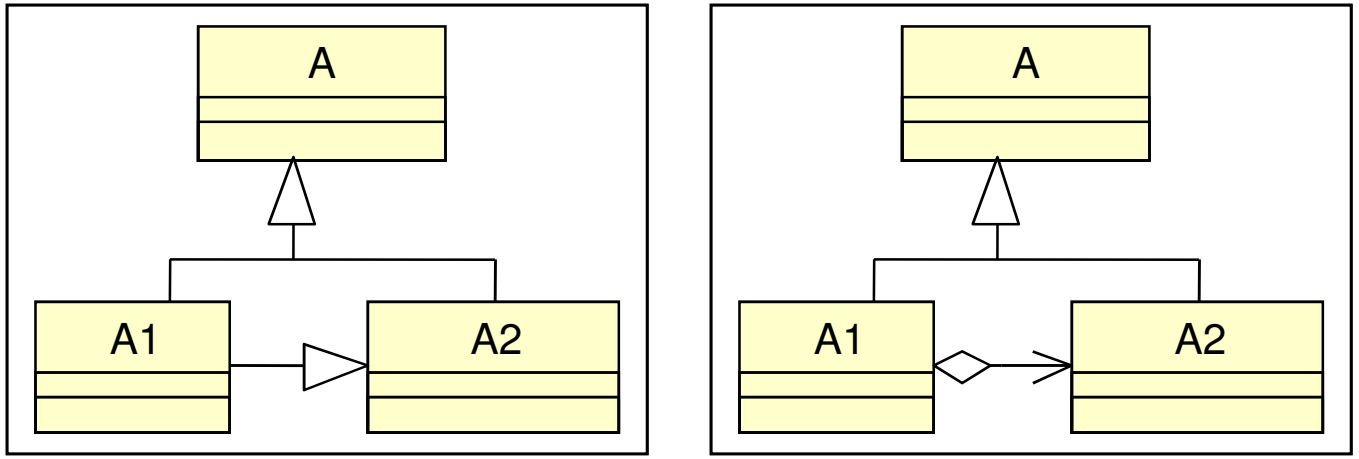
PRATIAN TECHNOLOGIES

Base Derived Collaboration

Collaboration	is-a	has-a
Auto	N	Y
Base Derived	Y	Y
Sibling		
Peer to peer		

PRATIAN TECHNOLOGIES

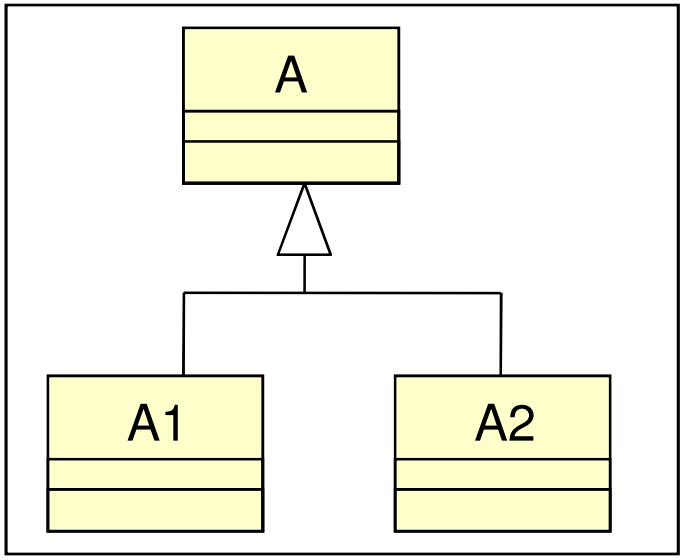
Sibling Collaboration



- Given that A1 has to reuse the implementation of A2, discuss which of the above two models is more appropriate

PRATIAN TECHNOLOGIES

Sibling Collaboration



- Given that the following relationships need to be achieved, how would you model the above classes

A1 can have A1	A2 can have A1
A1 can have A2	A2 can have A2

- Keep in mind maintainability should siblings collaborate with A3, A4, etc.

PRATIAN TECHNOLOGIES

Sibling Collaboration

Collaboration	is-a	has-a
Auto	N	Y
Base Derived	Y	Y
Sibling	N	Y
Peer to peer		

PRATIAN TECHNOLOGIES

Peer To Peer Collaboration

- A collaboration between two classes which realize different interfaces
- Can peers use ‘is-a’ or ‘has-a’

PRATIAN TECHNOLOGIES

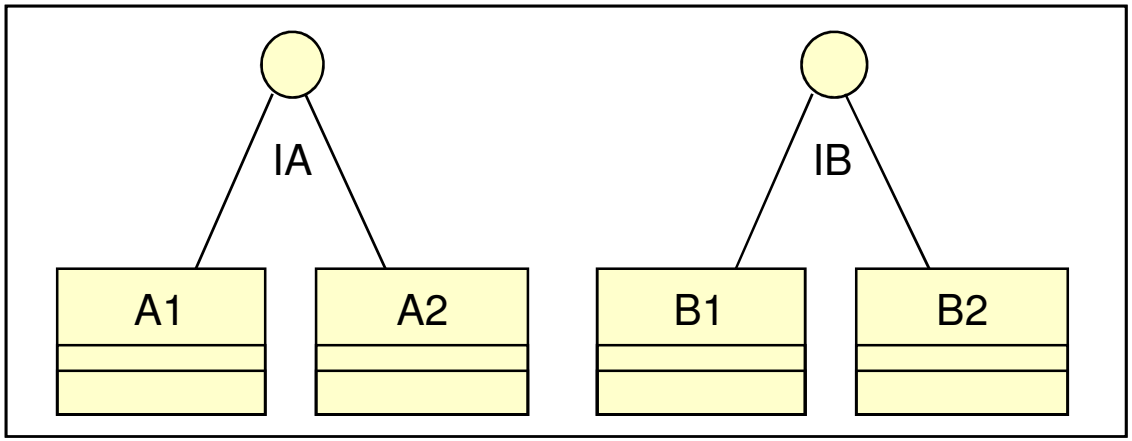
Peer To Peer Collaboration

Collaboration	is-a	has-a
Auto	N	Y
Base Derived	Y	Y
Sibling	N	Y
Peer to peer	N	Y

Prefer aggregation over inheritance

PRATIAN TECHNOLOGIES

Peer To Peer Collaboration



- Given that the following relationships need to be achieved, how would you model the above classes

A1 can have B1	B1 can have A1
A1 can have B2	B1 can have A2
A2 can have B1	B2 can have A1
A2 can have B2	B2 can have A2

PRATIAN
TECHNOLOGIES

Question Time

Please try to limit the questions to the topics discussed during the session.

Thank you.